2025-01-19 17:19:12

configs


```
{'_end': '2025-01-10:05',
 '_start': '2025-01-10:01',
 'exc': 'binance',
 'params': {'latencies': {'ack_private': 0,
                          'ack_public': 0,
                          'feed_private': 0,
                          'feed_public': 0,
                          'req_private': 0,
                          'req_public': 0},
            'maker_fees': 0.0,
            'taker_fees': 0.0},
 'tickers': ['SOLUSDT']}
```


fairprice metrics

		Q1(e%)	Q3(e%)	MEDIAN(e%)	MEAN(e%)	MSE(e%)	MAD(e%)	MSE*	MAD*
T+	Estimator								
t0	vamp100k	-0.003743	0.002893	-0.000325	-0.000447	0.000018	0.003568		
	vamp10k	0.000000	0.000000	0.000000	0.000001	0.000002	0.000679	*	*
	vamp200k	-0.004440	0.002503	-0.000950	-0.001001	0.000022	0.003854		
	vamp30k	-0.001454	0.001460	0.000000	-0.000041	0.000007	0.001949		
	vamp50k	-0.002563	0.002393	0.000000	-0.000140	0.000011	0.002774		
t1	vamp100k	-0.004179	0.003159	-0.000457	-0.000401	0.000060	0.005186		
	vamp10k	-0.000214	0.000000	0.000000	0.000047	0.000059	0.003743		*
	vamp200k	-0.004891	0.002762	-0.001088	-0.000955	0.000062	0.005395		
	vamp30k	-0.002253	0.001967	0.000000	0.000005	0.000057	0.004318	*	
	vamp50k	-0.003128	0.002764	-0.000100	-0.000094	0.000057	0.004751		
t10	vamp100k	-0.024027	0.023172	-0.000856	0.000551	0.001625	0.030386	*	*
	vamp10k	-0.023463	0.023515	0.000000	0.000999	0.001644	0.030497		
	vamp200k	-0.024568	0.022738	-0.001454	-0.000003	0.001626	0.030429		
	vamp30k	-0.023469	0.023481	-0.000109	0.000957	0.001632	0.030412		
	vamp50k	-0.023694	0.023518	-0.000620	0.000858	0.001627	0.030388		
t30	vamp100k	-0.045779	0.046394	0.000004	0.002506	0.005278	0.056574	*	*
	vamp10k	-0.045759	0.047617	0.000000	0.002955	0.005304	0.056697		
	vamp200k	-0.046414	0.045893	-0.000553	0.001953	0.005281	0.056607		
	vamp30k	-0.045464	0.046968	0.000000	0.002913	0.005289	0.056621		
	vamp50k	-0.045436	0.046695	0.000080	0.002814	0.005282	0.056590		
t60	vamp100k	-0.066359	0.075741	0.000708	0.005447	0.011199	0.083927	*	*
	vamp10k	-0.065029	0.075391	0.000000	0.005895	0.011219	0.084010		
	vamp200k	-0.067014	0.075139	0.000183	0.004893	0.011206	0.083975		
	vamp30k	-0.065611	0.075925	0.000915	0.005853	0.011206	0.083958		
	vamp50k	-0.065956	0.076140	0.001113	0.005754	0.011200	0.083934		
t120	vamp100k	-0.092908	0.111723	0.004120	0.011138	0.024199	0.122556		
	vamp10k	-0.092123	0.111577	0.005296	0.011586	0.024203	0.122519		
	vamp200k	-0.093495	0.111307	0.003602	0.010585	0.024222	0.122623		

	vamp30k	-0.092349	0.111730	0.005062	0.011544	0.024190	0.122501		*
	vamp50k	-0.092625	0.111929	0.004569	0.011446	0.024190	0.122518	*	
t300	vamp100k	-0.151786	0.182184	0.020085	0.027734	0.064511	0.200909		
	vamp10k	-0.151441	0.181571	0.021184	0.028182	0.064508	0.200859		
	vamp200k	-0.152522	0.181916	0.019570	0.027181	0.064566	0.201039		
	vamp30k	-0.151293	0.181780	0.021168	0.028140	0.064493	0.200841	*	*
	vamp50k	-0.151282	0.182020	0.020710	0.028041	0.064494	0.200856		

regression results

{'summary': <class 'statsmodels.iolib.summary.Summary'>

"""

OLS Regression Results

Dep. Variable:	b0	R-squared (uncentered):	0.003
Model:	OLS	Adj. R-squared (uncentered):	0.003
Method:	Least Squares	F-statistic:	110.5
Date:	Sun, 19 Jan 2025	Prob (F-statistic):	8.20e-26
Time:	17:19:11	Log-Likelihood:	1.8701e+05
No. Observations:	34426	AIC:	-3.740e+05
Df Residuals:	34425	BIC:	-3.740e+05
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
b1	1.4135	0.134	10.513	0.000	1.150	1.677

Omnibus:	80.864	Durbin-Watson:	0.014
Prob(Omnibus):	0.000	Jarque-Bera (JB):	81.384
Skew:	0.118	Prob(JB):	2.13e-18
Kurtosis:	3.036	Cond. No.	1.00

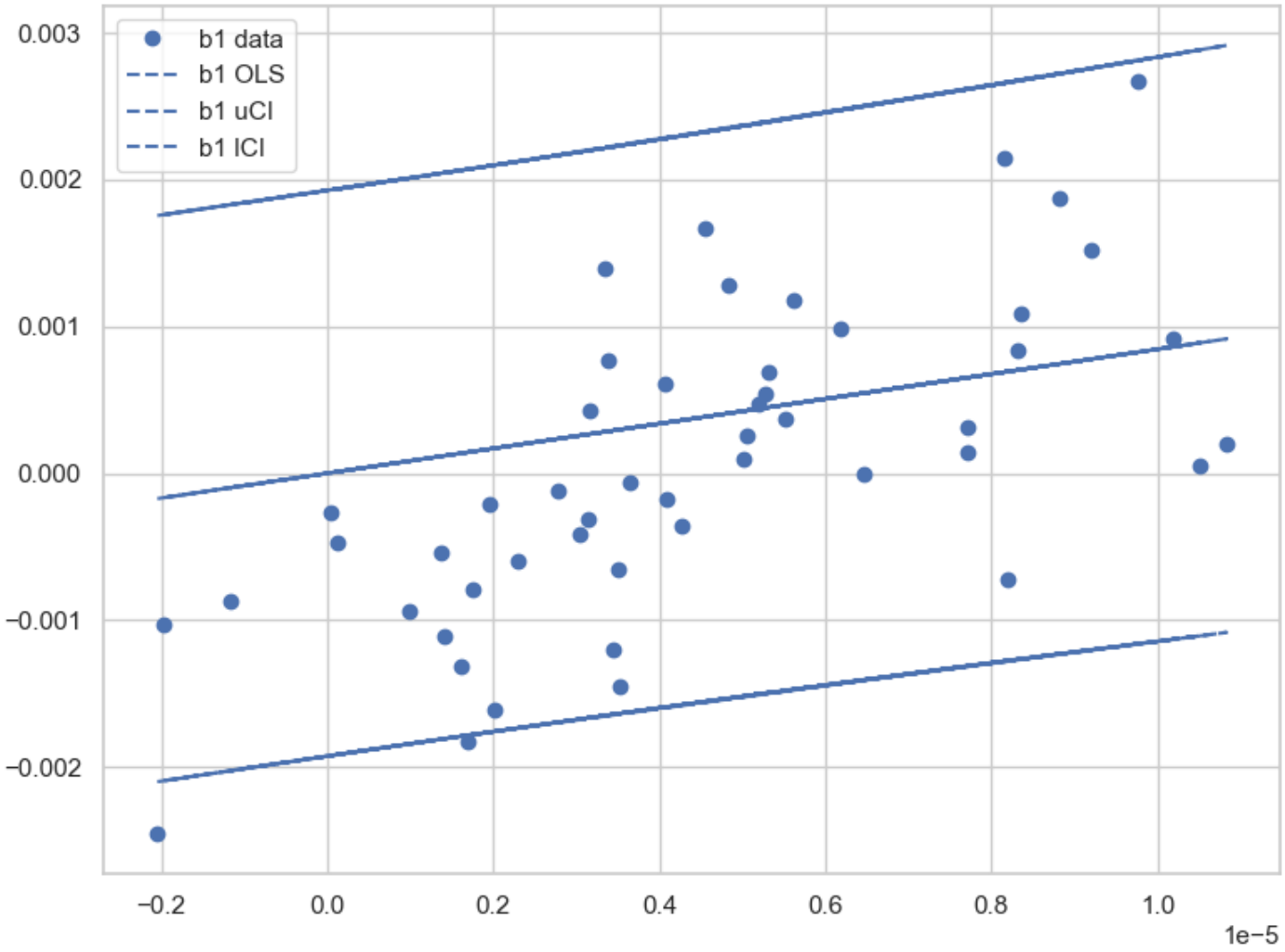
Notes:

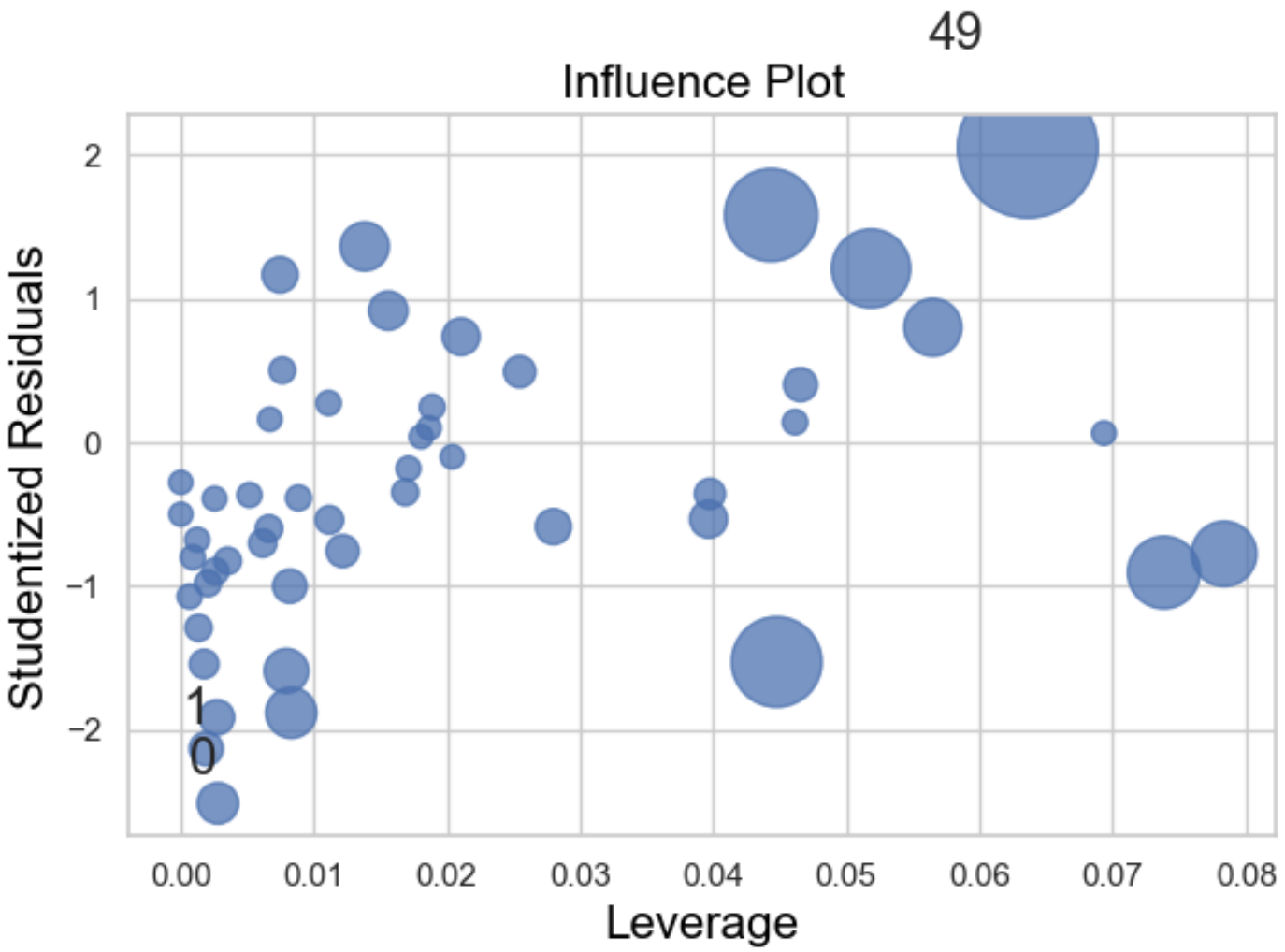
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.

[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

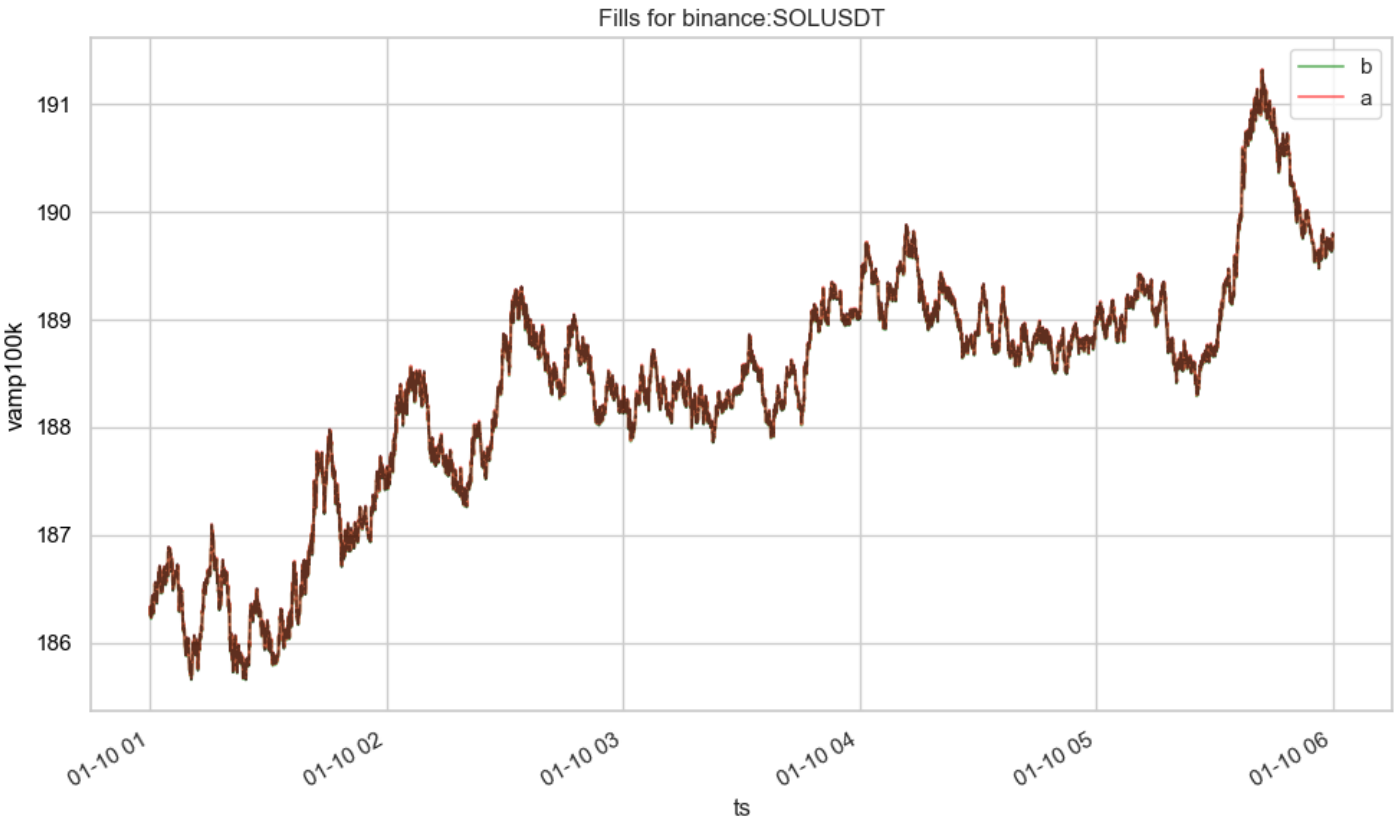
"""}

regression (binned)


b1 data
b1 OLS
b1 uCI
b1 lCI
0.003
0.002
0.001
0.000
−0.001
−0.002
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1e−5


49
Influence Plot
Studentized Residuals
Leverage
2
1
0
−1
−2
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
1
0

prices


Fills for binance:SOLUSDT
b
a
191
190
189
188
187
186
vamp100k
01-10 01
01-10 02
01-10 03
01-10 04
01-10 05
01-10 06
ts

Script

```
import os
import pytz
import asyncio
import logging
import argparse
import numpy as np
from pprint import pprint
from datetime import datetime
from dotenv import load_dotenv
load_dotenv()

from quantpylib.hft.oms import OMS
from quantpylib.hft.feed import Feed
from quantpylib.gateway.master import Gateway
from quantpylib.utilities.general import _time,pdf_snapshot

show = False
simulated = True

config_keys = {
    "binance":{},
}

exc = 'binance'
tickers = ["SOLUSDT"]
stream_data = {
    "binance":["SOLUSDT"],
}

run = None
time = None
replayer = None
print_interval = 120 * 1000

if simulated:
    from quantpylib.hft.mocks import Replayer,Latencies
    start='2025-01-10:01'
    end='2025-01-10:05'
    latencies={
        Latencies.REQ_PUBLIC:0,
        Latencies.REQ_PRIVATE:0,
        Latencies.ACK_PUBLIC:0,
        Latencies.ACK_PRIVATE:0,
        Latencies.FEED_PUBLIC:0,
        Latencies.FEED_PRIVATE:0,
    }
    replayer_configs = {
        "maker_fees":0.000,
        "taker_fees":0.000,
        "latencies":latencies,
    }
```

```
else:
    run = lambda : asyncio.sleep(1e9)
    time = lambda : _time()

gateway = Gateway(config_keys=config_keys)

async def quote_markets(replayer,oms,feed,ticker):

    last_print = time()
    async def l2_handler(lob):
        nonlocal last_print
        stamp = time()
        stamp = stamp - (stamp % print_interval)
        if stamp != last_print:
            print(datetime.fromtimestamp(time() / 1000,tz=pytz.utc))
            last_print = stamp

        for vamp in [10,30,50,100,200]:
            replayer.declare(
                key=f'vamp{vamp}k',
                value=lob.get_vamp(vamp * 1000),
                exc=exc,
                ticker=ticker,
                group='vamp'
            )

    l2_feed = await feed.add_l2_book_feed(
        exc=exc,
        ticker=ticker,
        depth=20,
        buffer=100,
        handler=l2_handler
    )

async def hft(replayer,oms,feed):
    await asyncio.gather(*[
        quote_markets(replayer=replayer,oms=oms,feed=feed,ticker=ticker)
        for ticker in tickers
    ])
    await run()

    if simulated:
        await sim_report(replayer)

async def sim_report(replayer):
    folder = './logs/reports'
    fairprices_plot = f'{folder}/fairprices.png'
    regression_plot = f'{folder}/regression.png'
    regression_influence = f'{folder}/regression_influence.png'
    price_plot = f'{folder}/prices.png'

    report_data = []
    report_data.append((
```

```
        'configs',
        {
            'exc':exc,
            'tickers':tickers,
            '_start':start,
            '_end':end,
            'params':replayer_configs,
        },
        'dict'
    ))

    metrics_df, declarations, error_dfs = replayer.df_fairprices(group='vamp',plot=False)
    report_data.append(('fairprice metrics',metrics_df,'df'))

    declarations = replayer.df_declarations(key='vamp100k',exc=exc,ticker='SOLUSDT',group='vamp')
    forward_prices = replayer.augment_forward_prices(declarations)

    from quantpylib.simulator.models import GeneticRegression
    model = GeneticRegression(
        formula='div(minus(t60,t0),t0) ~ div(minus(value,t0),t0)',
        intercept=False,
        df=forward_prices,
    )
    regression = model.ols()
    report_data.append(('regression results',{'summary':regression.summary()},'dict'))

    model.ols(bins=50)
    model.plot(fit=True,show=show,save_fit=regression_plot,save_influence=regression_influence)
    report_data.append(('regression (binned)',regression_plot,'img'))
    report_data.append(('regression (binned)',regression_influence,'img'))

    df_prices = replayer.df_prices(
        ticker='SOLUSDT',exc=exc,show=show,save=price_plot,
        key='vamp100k',group='vamp',share_index=True
    )
    report_data.append(('prices',price_plot,'img'))

    pdf_snapshot(
        save=f'{folder}/{datetime.now().strftime("%Y-%m-%d %H-%M-%S")}.pdf',
        code_file=__file__,
        report_data=report_data,
        include_comments=False
    )

async def sim_prepare():
    l2_data = {exchange:{} for exchange in stream_data}
    trade_data = {exchange:{} for exchange in stream_data}

    for exchange,tickers in stream_data.items():
        from utils import restore_archives
        await asyncio.gather(*[
            restore_archives(
                exc=exchange,
                ticker=ticker,
```

```
                depth=20,
                start=start,
                end=end,
            ) for ticker in tickers
        ])

        lob_archives = [
            Feed.load_lob_archives(
                exc=exchange,
                ticker=ticker,
                depth=20,
                start=start,
                end=end
            ) for ticker in tickers
        ]

        trade_archives = [
            Feed.load_trade_archives(
                exc=exchange,
                ticker=ticker,
                start=start,
                end=end
            ) for ticker in tickers
        ]

        l2_data[exchange] = {ticker:lob_archive for ticker,lob_archive in zip(tickers,lob_archives)}
        trade_data[exchange] = {ticker:trade_archive for ticker,trade_archive in zip(tickers,trade_archives)}

    global replayer, run, time
    replayer = Replayer(
        l2_data=l2_data,
        trade_data=trade_data,
        gateway=gateway,
        **replayer_configs
    )
    oms = replayer.get_oms()
    feed = replayer.get_feed()
    run = lambda : replayer.play()
    time = lambda : replayer.time()
    return oms, feed

async def main():
    await gateway.init_clients()
    if simulated:
        oms,feed = await sim_prepare()
    else:
        oms = OMS(gateway=gateway,exchanges=[exc],refresh_orders_snapshot=10,refresh_positions_snapshot=10)
        feed = Feed(gateway=gateway)

    await oms.init()
    await hft(replayer,oms,feed)
    await gateway.cleanup_clients()

if __name__ == '__main__':
```

```
asyncio.run(main())
```